**Penetration Testing Report**

**Full Name:**

**AYUSH V. PARKARA**

**Program: HCPT**

**Date: 21/02/2024**

# Introduction

This report document hereby describes the proceedings and results of a Black Box security assessment conducted against the **Week-1 Labs**. The report hereby lists the findings and corresponding best practice mitigation actions and recommendations.

# 1. Objective

The objective of the assessment was to uncover vulnerabilities in the **Week-1 Labs** and provide a final security assessment report comprising vulnerabilities, remediation strategy and recommendation guidelines to help mitigate the identified vulnerabilities and risks during the activity.

# 2. Scope

This section defines the scope and boundaries of the project.

|  |  |
| --- | --- |
| **Application**  **Name** | **Html Injection(HTML’s Are Easy!, Let Me Store Them!, Files Names Are also Vulnerable!, File Content and HTML Injection A Perfect Pair!, Injecting HTML Using URL, Encode IT ) , {Clickjacking (Let’s Hijack, ReHijack)}** |

# 3. Summary

Outlined is a Black Box Application Security assessment for the **Week 1 Labs**.

**Total number of Sub-labs: 8 Sub-labs**

|  |  |  |
| --- | --- | --- |
| **High** | **Medium** | **Low** |
| **1** | **3** | **4** |

**High - Number of Sub-labs with hard difficulty level**

**Medium - Number of Sub-labs with Medium difficulty level**

**Low - Number of Sub-labs with Easy difficulty level**

**1. Html Injection Labs**

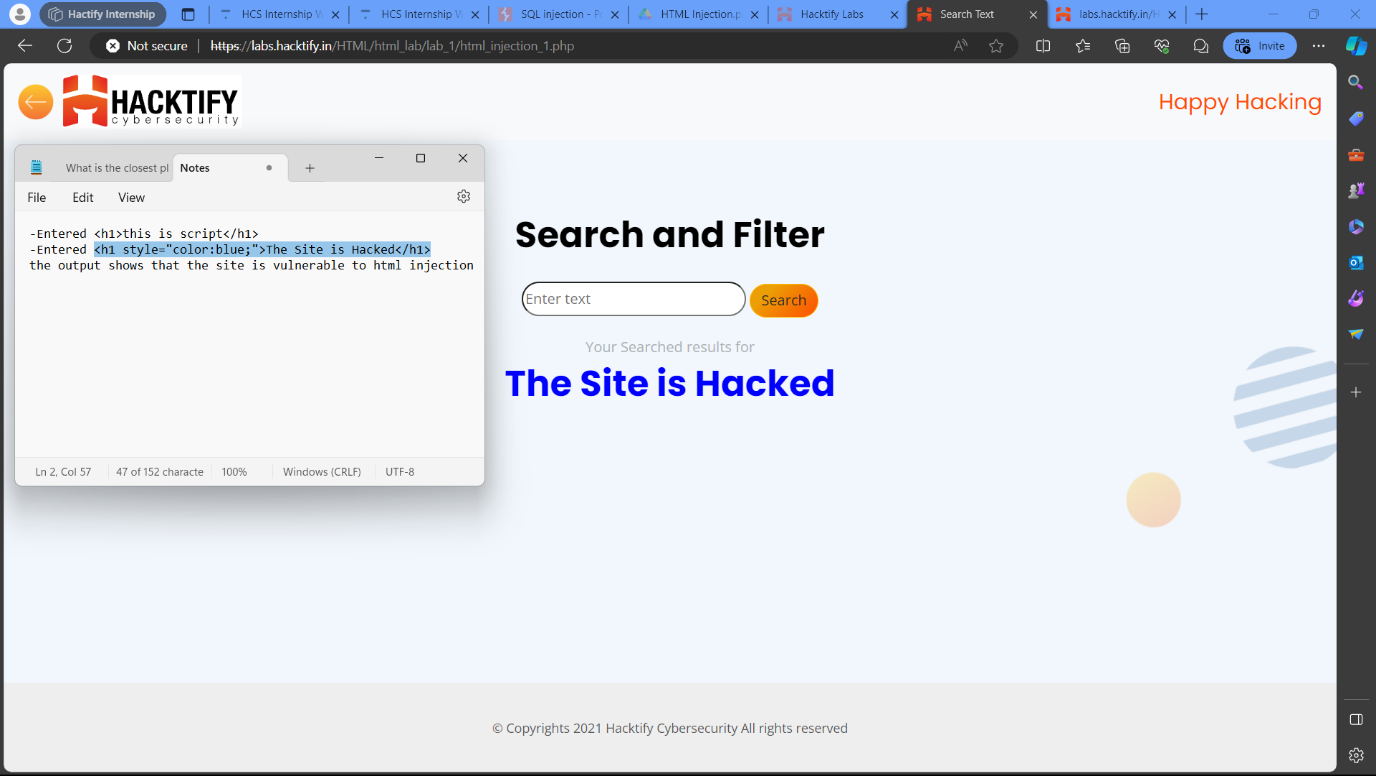
**1.1. HTML’s Are Easy!**

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| **HTML’s Are Easy!** | **Low** |
| **Tools Used** | |
| BROWSER(INSPECT) | |
| **Vulnerability Description** | |
| Inject a payload to check it and the result is that search parameter is vulnerable. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html\_lab/lab\_1/html\_injection\_1.php | |
| **Consequences of not Fixing the Issue** | |
| Data breach and identity theft, website defacement | |
| **Suggested Countermeasures** | |
| Input validation, output encoding, content-security-policy, parameterized queries. | |
| **References** | |
| https://www.softwaretestinghelp.com/html-injection-tutorial/ | |

# Observation and Application

* First of all after looking to the site we were able to see the search box and submit button.
* So then I saw the source code and inspected the inputs entered.
* Then by just normally applying the html injection normal <h1> text line got executed.
* By this way the first lab has got solved.
* We can also do more trial and error for the working tags like button and img are supported.
* But the italic and some similar is not supported also we cannot display the image even if uploaded.

# Proof of Concept



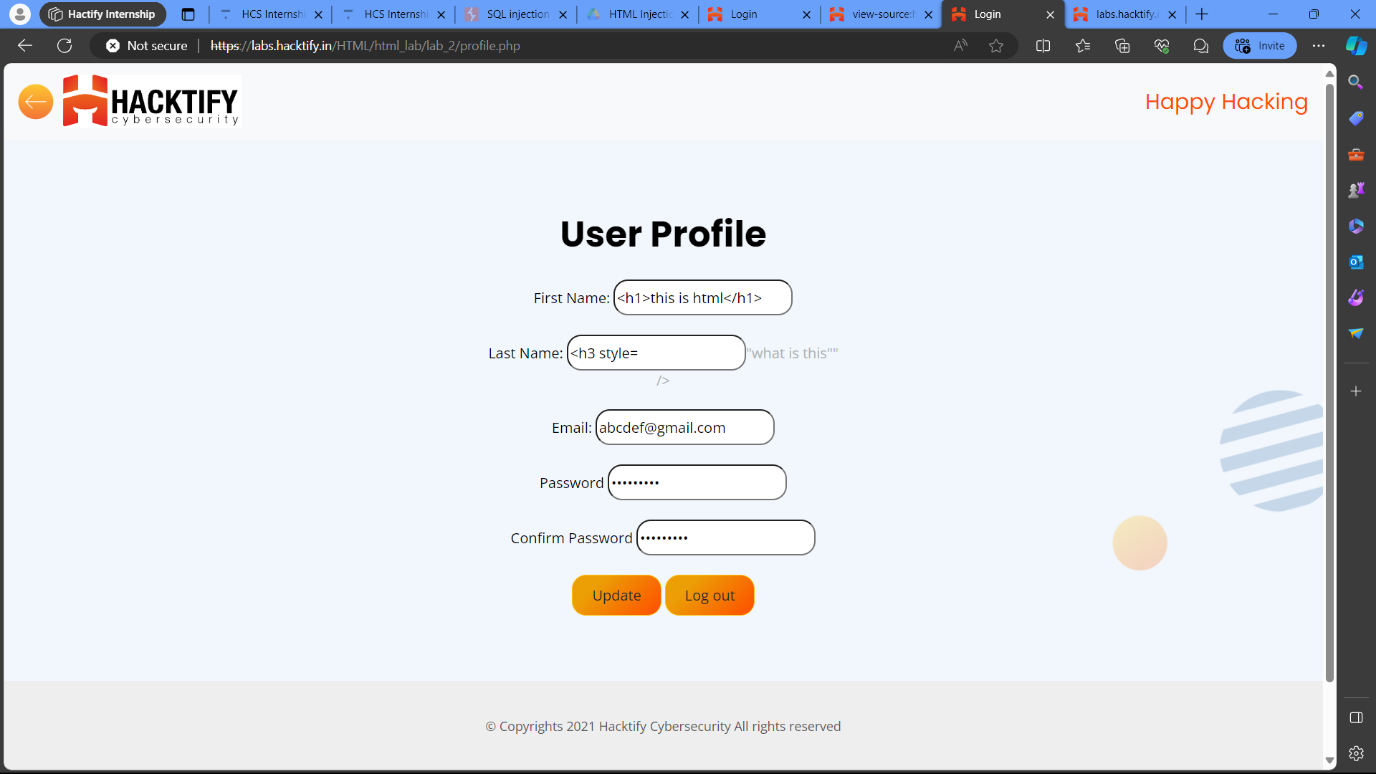
**1.2. Let me Store them!**

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| **Let me Store them!** | **Low** |
| **Tools Used** | |
| BROWSER(INSPECT AND SOURCE CODE) | |
| **Vulnerability Description** | |
| After observing payloads it’s persists as stored Html Injection | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html\_lab/lab\_2/html\_injection\_2.php | |
| **Consequences of not Fixing the Issue** | |
| For users: 1. Data theft, 2. Malware infection, 3. Phishing attacks; For website owner: 1. Website defacement, 2. Loss of user trust, 3. Legal and financial repercussion’s. | |
| **Suggested Countermeasures** | |
| Input validation, output escaping, regular update, security scanners. | |
| **References** | |
| https://www.softwaretestinghelp.com/html-injection-tutorial/ | |

# Observation and Application

* First of all after looking to the site we were able to see the User Profile page.
* In that I clicked login button without entering the data then also the page got redirected to update user details.
* But as no details were entered there was nothing shown inside text boxes.
* Then we did register with random data and then did the login process.
* It redirected with the data filled inside register form and then we can apply the html snippets and also can write the payload inside this.
* So that whenever the user logins the page the payload gets executed automatically as it got stored inside the source code or say user page.

# Proof of Concept



**1.3. File Names Are Also Vulnerable!**

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| **File Names are also Vulnerable!** | **Low** |
| **Tools Used** | |
| Burp suite. | |
| **Vulnerability Description** | |
| If a file name containing user input is incorporated into an HTML element without proper sanitization, malicious code embedded in the filename could be executed. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html\_lab/lab\_3/html\_injection\_3.php | |
| **Consequences of not Fixing the Issue** | |
| Execution content: 1. Data theft, 2. Defacement, 3. System compromise; non-executioncontent: 1. Social engineering, 2. File System Vulnerabilities | |
| **Suggested Countermeasures** | |
| Update software, Sanitize file name, Output escaping, input validation. | |
| **References** | |
| <https://www.softwaretestinghelp.com/html-injection-tutorial/> | |

# Observation and Application

* First of all after looking to the site we were able to see the Upload a file page.
* In this page I tried to change the url id but no results.
* But as it suggested from name I uploaded a file and then it reflected the file name.
* So by that we can get the idea that the we can write the payload inside the filename and then we can get it executed when we click the upload button.
* By this way one can enter payload to the site and also do data breaches and stealing as well as other things by just uploading payload and execute it using filename.

# Proof of Concept

# 

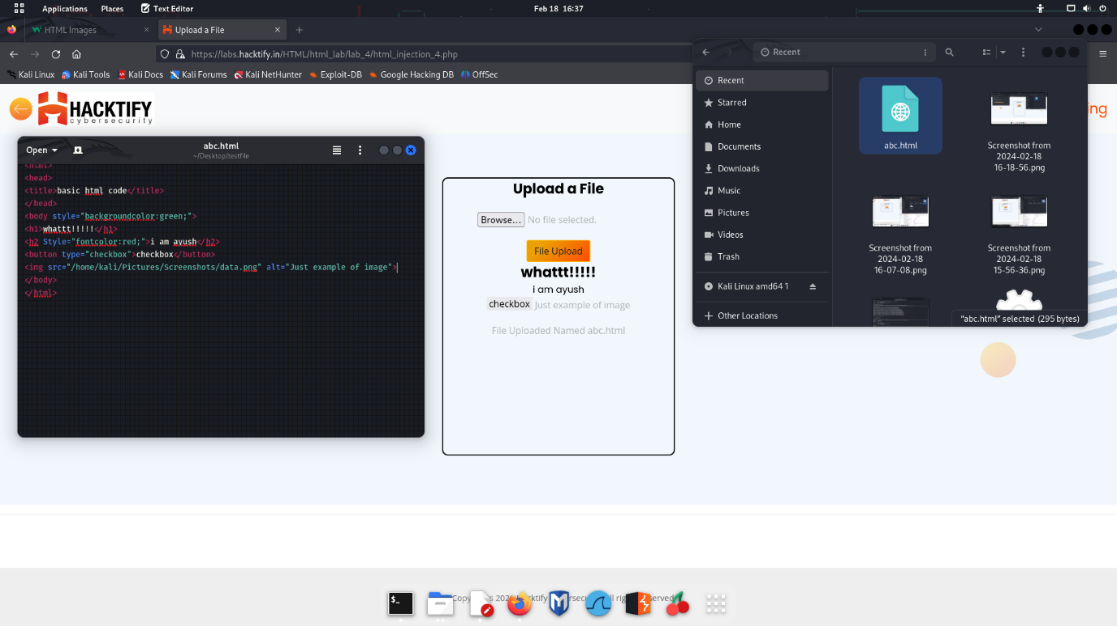
**1.4. File Content and HTML Injection a perfect pair!**

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| **File Content and HTML Injection a perfect pair!** | **Medium** |
| **Tools Used** | |
| Gedit(in linux) | |
| **Vulnerability Description** | |
| uploading a JavaScript file containing malicious code disguised as an image or document. When the website processes this file, the code gets executed, potentially leading to harmful actions. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| <https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php> | |
| **Consequences of not Fixing the Issue** | |
| What will be the consequences if the vulnerability is not patched? | |
| **Suggested Countermeasures** | |
| Regular security audits, validate and sanitize all file content, implement access control. | |
| **References** | |
| <https://www.softwaretestinghelp.com/html-injection-tutorial/> | |

# Observation and Application

* First of all after looking to the site we were able to see the same page as previous but there should be some difference.
* We won’t get same task in repeat so I searched for any changes.
* First of all I tried uploading a normal file and then saw that the file is being executed instead of file name as entering the same file as above uploaded.
* So then I used gedit in linux to type a normal html code and then saved the file using the extension ‘.html’.
* Then I tried uploading the file and then I got the result that the file is being executed instead of filename.
* So this one is actually bigger vulnerability ‘cause the site blindly executes the file that is being uploaded.
* Hence, one can easily upload any kind of payload and then execute it just by uploading it to this site and click upload.

# Proof of Concept



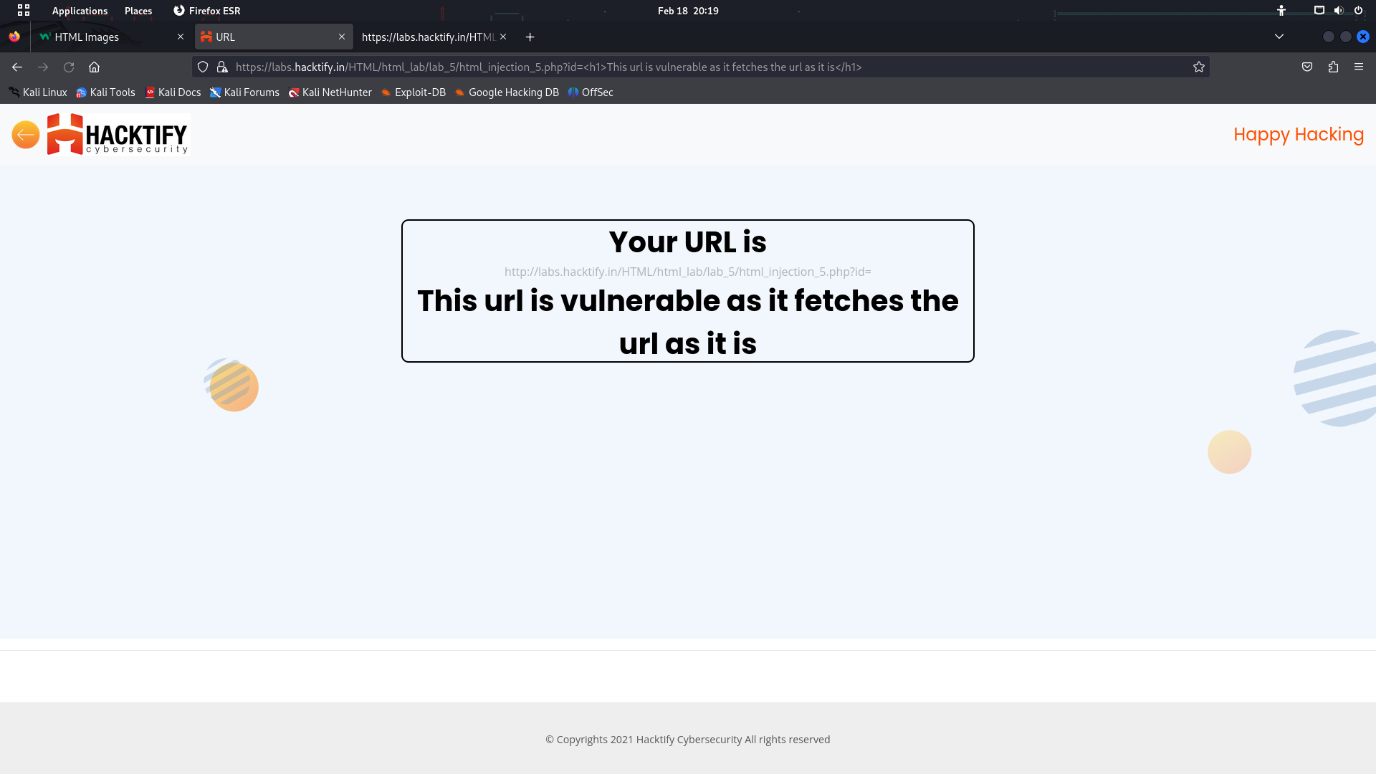
## 1.5. Injecting HTML using URL

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| **Injecting HTML using URL!** | **Medium** |
| **Tools Used** | |
| BROWSER(INSPECT AND SOURCE CODE) | |
| **Vulnerability Description** | |
| malicious code is inserted into the parameters of a website’s URL. This injected code can then be executed by the user’s browser when they access the URL, potentially leading to various harmful consequences. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| <https://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php> | |
| **Consequences of not Fixing the Issue** | |
| Data theft, phishing attacks, social engineering, hidden vulnerabilities. | |
| **Suggested Countermeasures** | |
| Validate and sanitize all user input, implement input validation libraries, perform regular security audits. | |
| **References** | |
| <https://www.softwaretestinghelp.com/html-injection-tutorial/> | |

# Observation and Application

* First of all after looking to the site we were able to see the webpage with the heading ‘Your URL is’ and in this page there is not interactive buttons or text.
* First of all by looking to source code it looked like the page is static nothing changes.
* But as it was displaying the webpage url I tried changing it and then refreshed it.
* So the url on the display changed so that we can get to know that the page shows the url that is currently active on the browser.
* Hence, we can write the script inside the url and we can execute it by doing the search.
* So for entering payload we have to add the keyword ‘?id=’ and then add the payload code and press enter.
* As the site gets refreshed the code or say payload gets executed and also the task is completed to find the vulnerability.
* We can add various type of payloads but the main thing will be trial and error, as the code may be supported or may be not for the browser.

# Proof of Concept



**1.6. Encode IT!**

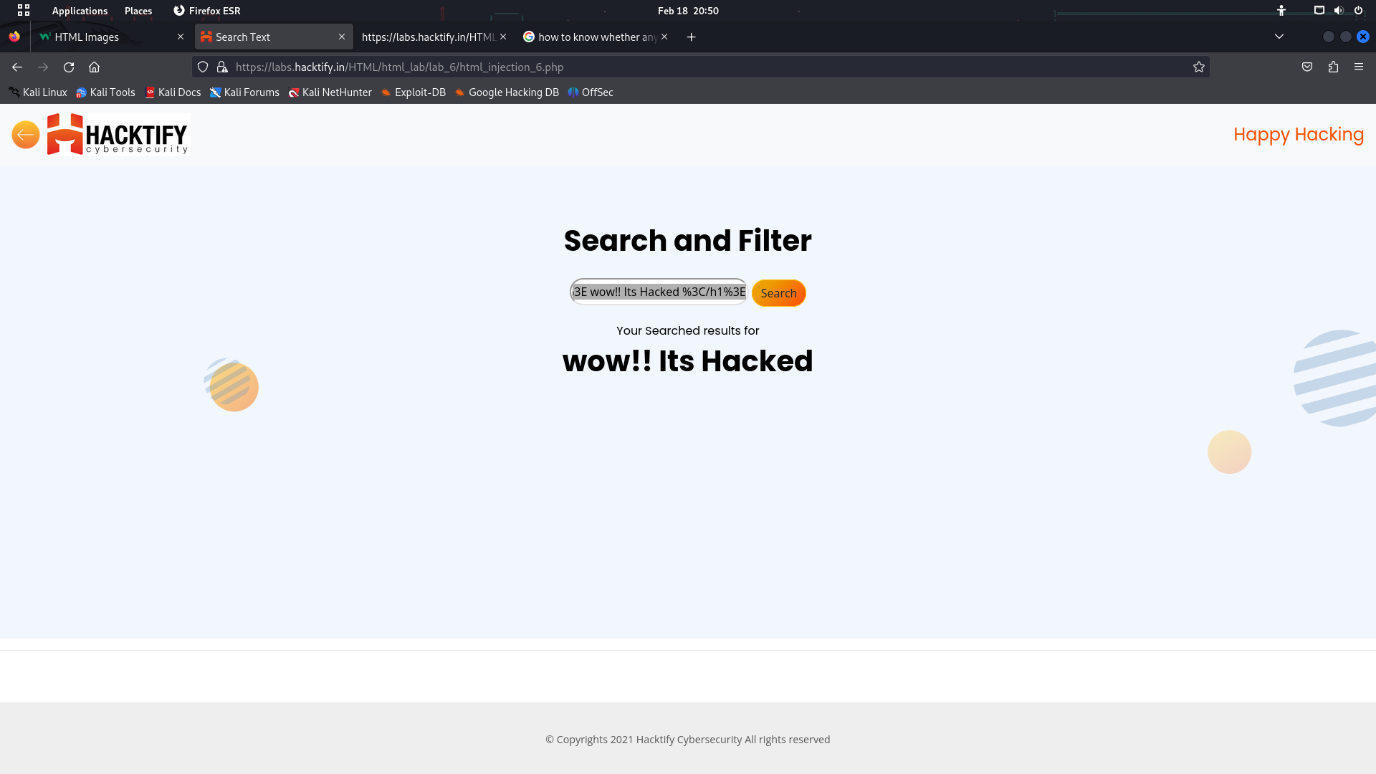
|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| **Encode IT!** | **High** |
| **Tools Used** | |
| HTML entity encoder. | |
| **Vulnerability Description** | |
| a **defense mechanism** used to transform user input before incorporating it into web pages. This transformation aims to **neutralize** any potentially harmful characters or code within the input that could be exploited by attackers | |
| **How It Was Discovered** | |
| Automated Tools / Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html\_lab/lab\_6/html\_injection\_6.php | |
| **Consequences of not Fixing the Issue** | |
| Data theft, social engineering, search engine ranking penalties, spreading of attack | |
| **Suggested Countermeasures** | |
| Always encode user input, combine with input validation, sanitize data, keep software updated | |
| **References** | |
| https://www.softwaretestinghelp.com/html-injection-tutorial/  <https://www.w3schools.com/tags/ref_urlencode.ASP> | |

# Observation and Application

* First of all looking at the webpage we can see that the webpage is somewhat like previous one in which we had something to search and then apply or perform search.
* But as said the level goes increasing so the webpage should not be repeated still I first tried the previous code used in previous similar kind of lab i.e. Lab 1.
* Now we got the new thing that the ‘<’ and ‘>’ bracket were replaced with ‘;’.
* This shows that the url has escape characters enabled.
* So first of all we have to try each and every possible value of these brackets one by one.
* So I have checked various values like binary code, ASCII value, and then I tried the encoding technique commonly used in which the table was publicly available with the values.
* Here the values used are taken from the site mentioned above which are as follows: ‘%3C’ used for ‘<’ and ‘%3E’ for ‘>’.
* We just have to replace these values instead of bracket and rest code can be as it is.
* Then I executed the search and bingo the code got executed.
* By these ways the current webpages are secured with much complex security parameters to prevent attacks.

# Proof of Concept

Injection used: %3Ch1%3E wow!! Its Hacked %3C/h1%3E



--------------------------------------------------------------------------------------------------------------

**2. Clickjacking Labs**

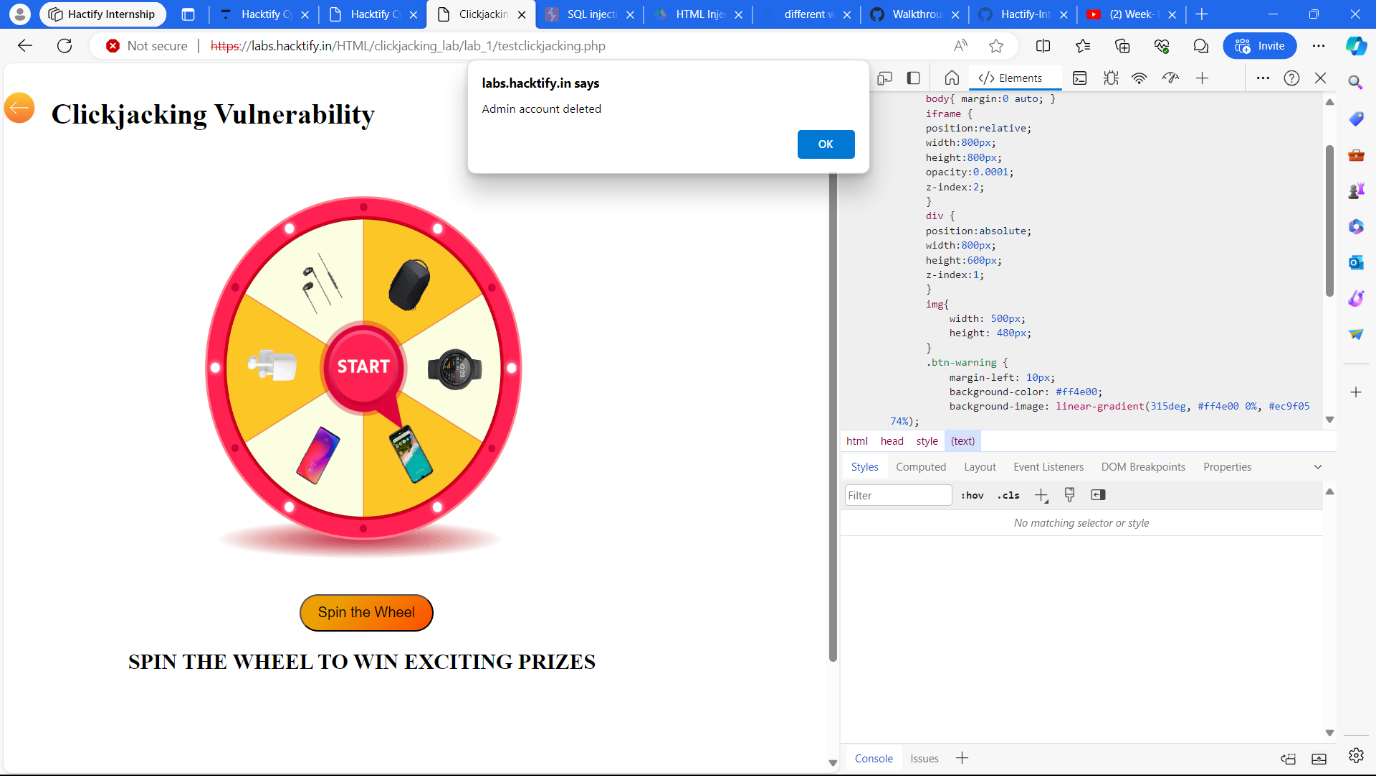
**2.1. Let’s Hijack!**

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| **Let’s Hijack!** | **Low** |
| **Tools Used** | |
| BROWSER(INSPECT AND SOURCE CODE) | |
| **Vulnerability Description** | |
| Clickjacking is a cyberattack that tricks user into clicking on hidden elements on a webpage. Attackers use invisible layers to overlay malicious buttons or links on legitimate websites, and when users click anywhere on the page, they unknowingly trigger the attacker's action. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/clickjacking\_lab/lab\_1/testclickjacking.php | |
| **Consequences of not Fixing the Issue** | |
| Malware download, financial losses, privacy violations, | |
| **Suggested Countermeasures** | |
| x-frame option header, CSP, Frame-busting JS code | |
| **References** | |
| https://owasp.org/www-community/attacks/Clickjacking | |

# Observation and Application

* First of all after looking to the site we were able to see that the original webpage was to perform the account deletion of admin or say the user logged in.
* Then after looking at the code and the inspection of the site I clicked the test button.
* Then appeared a new page with two pages overlapped one on another.
* The first page was the previous one that has the account logged on and then the next webpage was of spin the wheel in which the wheel was animated spinning and the spin the wheel button was overlapped on the ‘Delete Account’ button.
* So when we decrease the opacity to 0.0001 or less the overlapped page will only be seen i.e. the spin the wheel page and then when the user clicks the spin the wheel button.
* The user gets clickbaited and then the logged in account gets deleted.
* This is how clickbait and clickjacking works by overlapping or changing the task of the submit button.

# Proof of Concept



**2.2. Re-Hijack!**

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| **Re-Hijack!** | **Medium** |
| **Tools Used** | |
| BROWSER(INSPECT AND SOURCE CODE) | |
| **Vulnerability Description** | |
| Clickjacking is a cyberattack that tricks user into clicking on hidden elements on a webpage. Attackers use invisible layers to overlay malicious buttons or links on legitimate websites, and when users click anywhere on the page, they unknowingly trigger the attacker's action. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/clickjacking\_lab/lab\_2/testclickjacking.php | |
| **Consequences of not Fixing the Issue** | |
| Malware download, financial losses, privacy violations, | |
| **Suggested Countermeasures** | |
| x-frame option header, CSP, Frame-busting JS code | |
| **References** | |
| https://owasp.org/www-community/attacks/Clickjacking | |

# Observation and Application

* First of all after looking to the site we were able to see a login webpage.
* In which there is a gmail login page where we have to enter the gmail and password.
* After seeing the inspect and source code I was able to identify the doing of the webpage i.e. it will capture the data used or entered for the logging in the gmail.
* Then it will display at the screen also it can be made to send or store the data on the server unknowingly.
* So after this I clicked on the test button I saw two pages with gmail login where one was to login the gmail as well as one was made to reflect or say display as well as store the data fed to the page for logging in.
* By this way attacker can steal the login credentials as well as personal information from an individual.

# Proof of Concept

